Pergamon

Pattern Recognition, Vol. 29, No. 4, pp. 663–675, 1996
Elsevier Science Ltd

0031-3203/96 $15.00 + .00

0031-3203(95)00110-7

HAND-PRINTED ARABIC CHARACTER RECOGNITION SYSTEM USING AN ARTIFICIAL NETWORK

ADNAN AMIN,* HUMOUD AL-SADOUN† and STEPHEN FISCHER*

*School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia

†Department of Electrical and Computer Engineering, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

(*Received* 23 *January* 1995; *in revised form* 3 *July* 1995; *received for publication* 11 *August* 1995)

Abstract—Character recognition systems can contribute tremendously to the advancement of the automation process and can improve the interaction between man and machine in many applications, including office automation, check verification and a large variety of banking, business and data entry applications. The main theme of this paper is the automatic recognition of hand-printed Arabic characters using artificial neural networks in addition to conventional techniques. This approach has a number of advantages: it combines rule-based (structural) and classification tests; it is more efficient for large and complex sets such as Arabic characters; feature extraction is inexpensive and the execution time is independent of character font and size. The technique can be divided into three major steps. The first step is pre-processing in which the original image is transformed into a binary image utilizing a 600 dpi scanner and then thinned using a parallel thinning algorithm. Second, the image skeleton is traced from right to left in order to build a graph. Some primitives, such as Straight lines, Curves and Loops, are extracted from the graph. Finally, a five layer artificial neural network is used for the character classification The algorithm was implemented on a powerful MS-DOS microcomputer and written in C. The system was tested by 10 different users, whose writing ranged from acceptable to poor in quality and the correct recognition rate obtained was 92%.

Pattern recognition Arabic characters Hand-printed Parallel thinning
Feature extraction Structural classification Back-propagation Neural networks

1. INTRODUCTION

Character recognition is an area of pattern recognition that has undergone a great deal of research during the last 30 years. The recognition and processing of (hand) printed characters is motivated largely by a desire to improve man–machine communication.

Few products are currently commercially available for character recognition. Furthermore, their abilities are strictly limited to the recognition of typed text within a restricted number of fonts. Products to perform hand-printed text recognition are not available, although many approaches have been proposed. In fact there has recently been a high level of interest in applying artificial neural network architecture to solve this problem.[1–4]

Most of the development in neural networks research during the past decade made use of either pattern matching or statistical approaches for features extraction. Pattern matching is useful for printed characters but will not work for hand-written characters. A number of researchers have recently applied a neural networks technique to recognize the hand-printed characters ([5–7]) by using a statistical approach. However, these have generally been confined to relatively small character sets, usually digits only. This approach is more expensive and has not proven very efficient for large and complex sets.

Much more difficult, hence more interesting to researchers, is the ability to automatically recognize handwritten characters.[8–10] The complexity of the problem is greatly increased by noise and by the almost infinite variability of handwriting as a result of the mood of the writer and the nature of writing. Cursive script requires the segmentation of characters within the word and the detection of individual features. This is not a problem unique to computers; even human beings, who possess the most efficient optical reading devices (eyes), have difficulty in recognizing some cursive scripts and have an error rate of about 4% in reading tasks in the absence of a context.[11]

The different approaches covered under the general term character recognition fall into either on-line or off-line categories, each having its own hardware and recognition algorithms.

In on-line character recognition systems, the computer recognizes the symbols as they are drawn.[12–16] The most common writing surface is the digitizing

tablet, which has a resolution of 200 points per inch and a sampling rate of 100 points per s and deals with one dimensional data.

Off-line recognition is performed after the writing or printing is completed. Optical Character Recognition OCR[17-21] deals with the recognition of optically processed characters rather than magnetically processed ones. In a typical OCR system, input characters are read and digitized by an optical scanner. Each character is then located and segmented and the resultant matrix is fed into a pre-processor for smoothing, noise reduction and size normalization. Off-line recognition can be considered the most general case: no special device is required for writing and "signal" interpretation is independent from signal generation, as in human recognition.

Many papers have been concerned with Latin, Chinese and Japanese characters. However, although Arabic characters are used in writing several widespread languages, little research has been conducted toward the automatic recognition of Arabic characters because of the strong cursive nature of its writing rules.

The IRAC (Interactive Recognition of Arabic Character) system proposed by Amin *et al.*[22] adopts a structural classification method for recognizing on-line hand-written isolated Arabic characters. Features such as the shape of the main stroke, the number of strokes, the number and position of a group of dots and the presence of an eventual "zigzag" are then extracted from the character. If these features do not permit the recognition of a character by a simple dichotomic consultation of the dictionary, a backtracking is performed in order to correct the shape of the main stroke. Finally, secondary features of the main stroke such as the frame size, the start point and the curvature are extracted using a distance function in order to remove the ambiguity and provide an exact match.

In reference (23), a system for recognition of cursive Arabic words was discussed . Words are entered via a graphic tablet and segmented into characters. Characters are then recognized using a method similar to that for isolated characters.[22] Finally, word recognition works by constructing all possible words by following every path in the equivalence graph of the lattice. Binary diagrams[24] are also used to discard ineligible combination of letters.

Amin *et al.*[24] introduced two methods for recognizing on-line cursive Arabic words. The first is a syntactical method based on the segmentation of words into primitives such as curves and strokes. An automaton transforms the primitives into a list of the characters constituting the word. The second method uses a global approach: each word is identified according to a vector of some pre-determined parameters. Furthermore, to enhance the recognition rate a syntactic and semantic analyser that verifies the grammatical structure and the meaning of Arabic sentence is used.

El-Sheikh and El-Taweel[26] proposed a system for recognizing properly segmented hand-written Arabic characters. Four groups are identified, depending on the position of the character within the word (isolated, beginning, middle or at the end). Moreover, each group is further classified into four subgroups depending on the number of the strokes (one, two, three or four) in the character. The classification of characters within each subgroup is done by means of some features, such as having a maximum or a minimum on either horizontal or vertical direction, the ratio between length and width, the type of secondary stroke and other features.

Al-Emami and Usher[27] presented a system for on-line recognition of hand-written Arabic words. Words are segmented into strokes based on the method in reference (28). In the learning process specifications of the strokes of each character are fed to the system, while in the recognition process the parameters of each stroke are found and special rules are applied to select the collection of strokes that best match the features of one of the stored characters. However, few words were used in the learning and testing processes, which makes the efficiency and accuracy of the system questionable.

Parhami and Taraghi[29] introduced an off-line technique for the automatic recognition of printed Farsi text (which is similar to Arabic text). The authors selected 20 features based on certain geometric properties of the Farsi symbols to construct a 24-bit vector and then compare it with entries in a table where an exact match is checked first. The system is heavily font-dependent and the segmentation process is expected to give incorrect results in some cases.

Table look-up is used for the recognition of isolated hand-written Arabic characters.[30] In this approach, the character is placed in a frame which is divided into six rectangles and a contour tracing algorithm is used for coding the contour as a set of directional vectors by using a Freeman code. However, this information is not sufficient to determine Arabic characters. Therefore, extra information related to number of dots and their position is added. If there is no match, the system will add the feature vector to the table and consider that character as a new entry.

Amin and Masini[31] adopted a structural approach for recognizing printed Arabic text. Words and subwords are segmented into characters using the baseline technique. Features such as vertical and horizontal bars are then extracted from the character using horizontal and vertical projections. Four decision trees according to position of the character within the word are used and the structure of these decision trees allows a rapid search for the appropriate character. Furthermore, trees are utilized to distinguish characters that have the same shape but appear in different positions within a word.

Amin and Mari[32] proposed a new technique for multi-font Arabic test. The system segments words into characters using a vertical histogram with some certain rules that have to be satisfied to insure the correctness of segmentation. This is followed by scan-

ning the character through a 2 × 2 window for contour detection; a Freeman code[33] is used for character description. Finally, a Viterbi algorithm, which is based upon a probabilistic approach,[34] is used for Arabic word recognition to enhance the recognition rate. The main advantage of this technique is that it allows the use of an automatic learning process.

Almuallim and Yamaguchi[35] presented a structural recognition technique for Arabic hand-written words. Their system consists of four phases. The first is pre-processing, in which the word is thinned and the middle of the word is calculated. Since it is difficult to segment a cursive word into letters, words are then segmented into separate strokes and classified as strokes with a loop, strokes without a loop and complementary characters. These strokes are then further classified using their geometrical and topological properties. Finally, the relative position of the classified strokes are examined and the strokes are combined in several steps into the string of characters that represents the recognized word. Most errors were due to incorrect segmentation of words.

El-Sheikh and Guindi[36] introduced a system for recognizing typewritten Arabic text. Words are segmented by tracing the outer contour and calculating the distance between the extreme points of intersection of the contour with a vertical line. The segmentation is based on a horizontal scan from right to left of the closed contour using a window of adjustable width. For each position of the window, the average vertical distance is calculated across the window. Finally, a set of Fourier descriptors derived from the coordinate sequences of the character outer contour is used to distinguish the different characters.

Khaly and Sid-Ahmed[37] utilized moment invariant descriptors to recognize the segmented characters, while El-Dabi *et al.*[38] adopted a statistical approach for recognizing Arabic typewritten characters. In this approach the characters are segmented and recognized by using accumulative invariant moments as feature descriptors. This system is font-dependent and sensitive to small variations in input characters.

Al-Yousefi and Uda[39] introduced a statistical approach for recognizing Arabic characters. The character is segmented into primary and secondary parts. Secondary parts of a character are then isolated and identified separately and the moments of horizontal and vertical projections of the primary part of the character are computed. The system requires a long processing time.

Shoukry[40] used a sequential algorithm based on the input-time tracing principle which depends on the connectivity properties of the acquired text in the image binary domain. This algorithm bears some resemblance to an algorithm devised by Wakayama[41] for the skeletonization of binary pictures.

The SARAT system[42] uses outer contours to segment an Arabic word into characters. The word is divided into a series of the curves by determining the start and end points of the word. Whenever the outer contour changes sign (starting with a positive to a negative curvature) a character is segmented.

Recently, Al-Sadoun and Amin[43] adopted a structural technique for segmenting and recognizing Arabic printed text. The algorithm can be applied to any font and it permits the overlaying of characters. The technique can be divided into five major steps. First, the digitization in which the original image is transformed into a binary image utilizing a scanner (300 dpi). Second is the pre-processing step, in which a parallel thinning algorithm is applied to the Arabic binary word. Third, the image skeleton is traced from right to left using a 3 × 3 window and a binary tree is constructed. The Freeman code is used to describe the skeleton shape. Fourth, the binary tree is segmented into sub-trees such that each sub-tree describes a character in the image. Finally, the character description is transferred from a binary tree structure to a string of primitives. This string is used to identify the character utilizing four decision trees.

This paper proposes a new technique for the recognition of hand-printed Arabic characters using neural networks. There are four main advantages of this technique. First, the proposed technique combines rule-based (structural) and classification tests. Second, it is more efficient for large and complex sets, such as Arabic characters. Third, feature extraction is inexpensive. Fourth, the execution time is independent of both the character font and size. This paper describes the neural network method applied in the classification step, the computation intensive earlier stages being carried out by more classical approaches. Thus, this system takes a hybrid approach to character recognition.

The paper is organized as follows: Section 2 reviews some of the basic features of the Arabic text; Section 3 outlines the procedure for digitization and pre-processing of the image; Section 4 introduces the features of the image and the algorithm to trace it; Section 5 outlines the algorithms to extract the features from the image; Section 6 introduces the character classification technique using artificial neural networks; Section 7 presents the results of the experiments performed; and finally, some concluding remarks are made in Section 8.

2. GENERAL CHARACTERISTICS OF ARABIC CHARACTERS

Arabic, like Hebrew, is written from right to left. Arabic text (printed or hand-written) is cursive in general (i.e. Arabic letters are normally connected on the base line). Therefore, the recognition rate of Arabic characters is lower than that of disconnected characters such as printed English.

Arabic writing is similar to English in that it uses letters, numerals, punctuation marks, as well as space and special symbols (e.g. mathematical notation). However, it differs in its representation of vowels since Arabic utilizes various diacritic markings. Diacritics take on the form of over- and underscores of letters.

(Note however that most current written Arabic texts are unvocalized. This study therefore assumes that the text is unvocalized). In addition, the structure of an Arabic character consists of curves and line segments. Many Arabic characters contain a loop, while others contain double loops.

The Arabic alphabet is represented numerically by a standard communication interchange code approved by the Arab Standard and Metrology Organization (ASMO). Similar to the American Standard Code for Information Interchange (ASCII), each character in the ASMO code is represented by one byte. The English letter has two possible shapes. One is called 'the Capital shape', while the other is termed 'the Small shape'. The ASCII code provides separate representations for both of these shapes. An Arabic letter has only one representation in the ASMO table. This is not to say that the Arabic letter has only one shape. On the contrary, an Arabic letter might have up to four shapes depending on its relative position in the text. For instance, the letter "A'" in ε" has four shapes: at the beginning of the word (preceded by a space); in the middle of the word (no space around it); at the end of the word (followed by a space) and in isolation (preceded by an unconnected letter and followed by space). These four possibilities are shown in Fig. 1.

In addition, some Arabic characters have exactly the same shapes, which are distinguished from each other only by the addition of a complementary character. These complementary characters include a single dot, two dots, three dots, a zig-zag and a vertical bar shape. Complementary characters are positioned either above, below or within the confines of the character. Figure 2 depicts two sets of characters, the first set having five characters and the other set having three characters. Each set contains characters which differ only by the position and/or the number of dots associated with it.

3. DIGITIZATION AND PRE-PROCESSING

Extensive work has been conducted on the binarization process.[44,45] The binarization algorithm employed in this work is similar to the method appearing in reference (46). A 600 dpi scanner is used to digitize the image in this phase and the output is a standard binary formuatted image (PBM format).

The pre-processing step is intended to reduce noise and increase the ease of extracting structural features from the digitized image. This step involves cleaning and thinning the image.

3.1. *Pre-thinning*

This step aims to reduce the noise that binarization process yields. The pre-thinning algorithm used in this study is as follows:

Input: A digitized image I in PBM format.
Output: A pre-thinned image I' in PBM format.
begin

1. For each pixel P in image I, let P0 to P7 be its 8 neighbors, starting from the east neighbor and counted in anti-clockwise fashion:

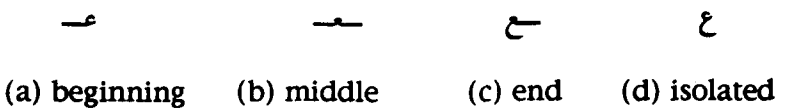
   P3 P2 P1
   P4 P P0
   P5 P6 P7

2. Let B(P) = P0 + P2 + P4 + P6
   Let P' be the pixel in I' corresponding to P.
3. IF B(P) < 2 THEN Set P' to white
   ELSIF B(P) > 2 THEN Set P' to black
   ELSE Set P' to the value of P;

end

3.2. *Thinning*

Many thinning algorithms are proposed in the literature. These algorithms can be classified as either



(a) beginning (b) middle (c) end (d) isolated

Fig. 1. Different shapes of the Arabic letter "A" in ε.

(a) (b)

Fig. 2. Arabic characters differing only with regard to the position and number of associate dots.

parallel or sequential algorithms. Parallel algorithms thin the entire image simultaneously, while sequential algorithms iteratively process the image. A parallel algorithm based on Jang and Chin[47] is used in this study. Figure 3 shows an original scanned image and the resulting skeleton after applying the thinning algorithm.

4. FEATURE PRE-PROCESSING

4.1. *Feature point extraction*

A feature point is a black pixel in the thinned image which has a number of black neighbors not equal to 0 or 2. The types of feature points are End points, Branch points and Cross points. An End point, as its name suggests, is the end/start of a line segment, a Branch point connects three branches, while a Cross point connects four branches. Figure 4 depicts these three types.

The number of branches connected to a feature point is used to determine the connection of nodes in skeleton tracing. The traditional method of detecting Branch points is by counting the number of black neighbors in a 3 × 3 pixel window. Instead of using this method, the algorithm uses a new method to detect and classify a feature point. The algorithm counts the number of transitions from black to white pixels in the surrounding 3 × 3 window in order to determine the type of feature point. This method does not have the problem of multiply defined Branch or Cross points. To illustrate this technique Fig. 5 shows three points: a, b and c. If the sum of the black neighbors method is used then points a, b and c will have a value of 3. Therefore, all these points will be defined as Branch points. On the other hand, if the number of transitions method is used then only point b will be defined as a feature point and classified as a Branch point.

The feature points detection algorithm horizontally scans the input image pixel by pixel and obtains the corresponding 3 × 3 window for each pixel. The algorithm scans the window boundary for pixel P starting from the East pixel (P0) and follows a clockwise direction; the number of transitions is then calculated. If the number of crossings is one, this point is classified as an End point and its coordinate is then recorded. Similarly, the number of crossings is three for a Branch point

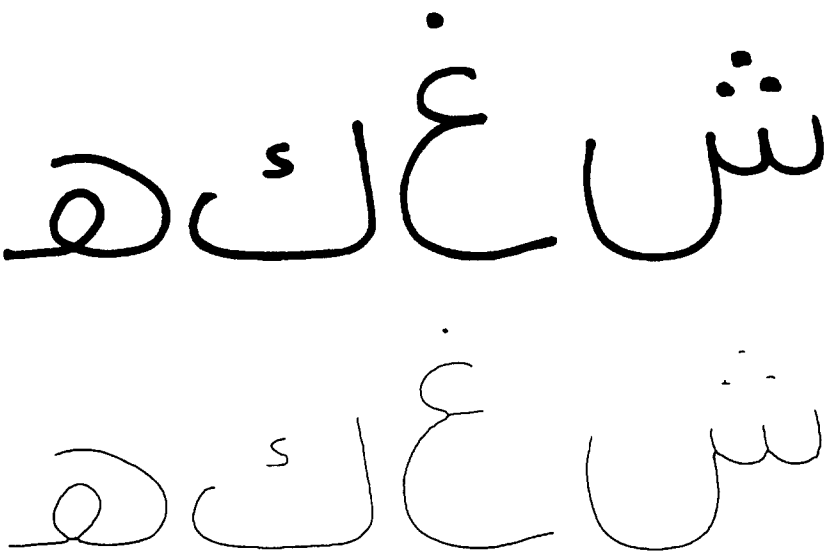


Fig. 3. An example for the Arabic characters before and after thinning.

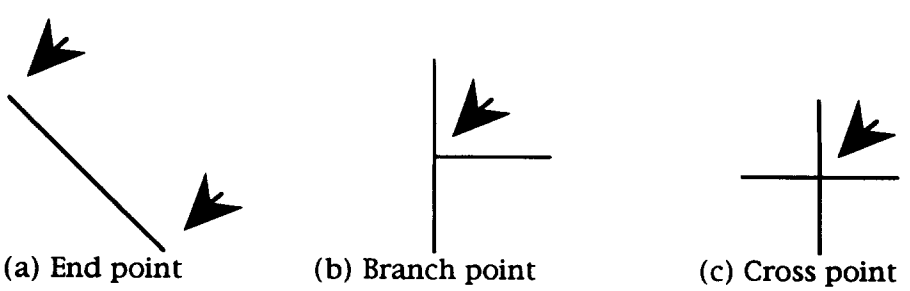


Fig. 4. The types of feature point.

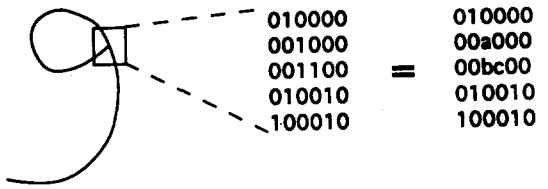


Fig. 5. Difference between using sum of neighbor and number of transitions methods.

and four for a Cross point. Figure 6 depicts this algorithm.

The position attribute of a feature point has three possible values: Upper, Middle or Lower. This attribute represents the relative position of the feature point within the image.

4.2. *Skeleton tracing*

The next stage of the feature extraction process is to build a graph representation of the input image. The data structure used is an adjacency matrix. The maximum number of a particular type of feature point that can exist in an Arabic character is 5. Therefore, the first five indices of the matrix correspond to End points, the next five indices correspond to Branch points and the last five indices correspond to Cross points. An extra index is included in the matrix data structure and serves as a flag to determine an input image that has no feature point.

The algorithm traces the thinned character and creates a 16 × 16 matrix to represent the input character. Each element $[i][j]$ of the matrix contains four fields of data:

(a) Number of connection lines from feature point i to feature point j. Hence, this field contains the adjacency information of the graph. This field is the same for element $[i][j]$ and element $[j][i]$. This field will be 1 in element $[i][j]$ in the case of a loop.

(b) A freeman code[33] string describing the line connecting point i to point j and its length. The values of this field in elements $[i][j]$ and $[j][i]$ will be complements. However, the lengths will be the same.

(c) The maximum and minimum values of the x and y coordinates of the line connecting point i to point j. This field is the same for element $[i][j]$ and element $[j][i]$. These values are used for determining the position and area information that will be used in later stages of the algorithm.

(d) An optional pointer that points to another element in order to describe another line connecting point i to point j. If the number of connections between the two points is 1, this pointer will be null.

It is worth noting here that this image tracing algorithm is basically a depth first-search algorithm. The algorithm is described below:

```
variable: S:Stack;
variable: start, end:Point;
variable: flag:Boolean;
{
  if a feature point exists then pick one
  else Choose the first black pixel encountered in the
image
Push the picked point onto S
  while S is not empty do
    start:= Pop S
    flag:= TRUE
    while (flag) do
      Get 3 x 3 window of start
      if (number of black neighbors of start = O)
        then flag:= FALSE
      if (number of black neighbors of start > 1)
        then Push start to S
      Trace and delete from start to next feature
      point (end)
      Update Matrix [start][end]
      if (Matrix[start][end]. 1st_Field > 0)
        then Allocate next pointer
      if (start ≠ end) then Update Matrix [end][start]
      start:= end
    end
  end
}
```

The trace and delete step in the above algorithm will erase the traced branch so that it will not be traversed again. Special care must be taken to implement such

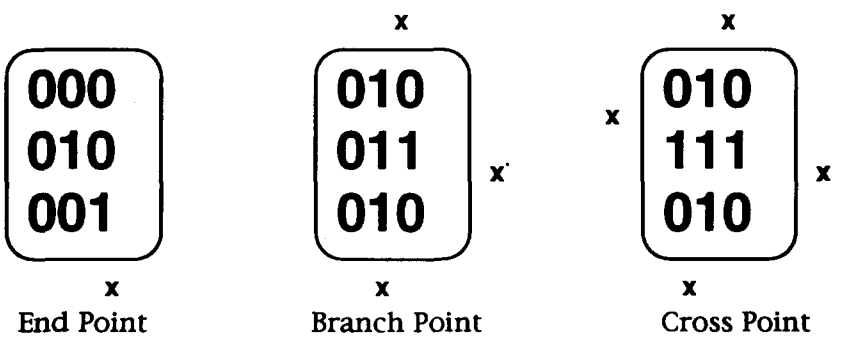


Fig. 6. Number of transitions algorithm for detecting the feature point (x represents a boundary crossing).

a function when going through a turning point or near another feature point. These cases will be explained in the following paragraph.

The *Trace and Delete function* utilizes a 3 × 3 window to trace from a starting point until it reaches a feature point or the original starting point. During tracing, it generates a linked list of directional codes. In addition, it records the maximum and minimum coordinates of the traced segment. Traced pixels are deleted to prevent back tracing. One special case is when reaching a turning point, from which multiple directions are reachable, as depicted in Fig. 7. In order to solve this problem, eight different window priority templates are used so that the trace will follow the previous moment of direction, as shown in Fig. 8. In these priority templates the highest priority is given to the trace's previous direction. Thereafter, higher priority is given to the ones that are closer to the previous direction. For example, in Fig. 7 the algorithm traces from a to b and has to make the decision whether go to c or d next. According to the priority template, it will choose to go to c because it has the same trace direction as the one from point a to point b.

Another special case that the *Trace and Delete function* has to take care of is the pixel just before reaching a feature point. This case is demonstrated in Fig. 5, points a, b and d. Since the number of black neighbors at the pixel window will be greater than one, decision has to be made in order to reach the destination feature point. The solution is a one-pixel-lookahead checking; for each reachable black pixel in the 3 × 3 window, the algorithm checks whether the pixel is a feature point or not. If it is, the trace goes in its direction. The lookahead checking also follows the priority templates, depicted in Fig. 8, so that the tracing always follows the same direction moment.

For multiply segmented characters, we can repeat the skeleton tracing until all segments in the character image are deleted.

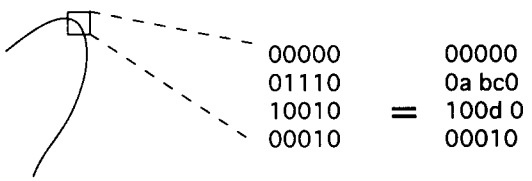


Fig. 7. Turning point illustration.

5. FEATURE EXTRACTION

5.1. *Complementary characters extraction*

As mentioned in Section 2, some Arabic characters are accompanied by complementary characters. These complementary characters are dots and hamza (a zigzag shape).

The number of dots and their location relative to the main skeleton of the character have to be identified. The number of dots can be one, two or three. The dots can be below or above the main skeleton of the character.

The complementary characters can be identified easily from the 16 × 16 matrix formed earlier. These characters are small in size relative to the overall image. Furthermore, there are few feature points within them.

5.2. *Loop extraction*

5.2.1. *Loop positioning.* Three types of loops are defined: Large, Small Upper and Small Lower. A Large loop occupies nearly all of the input image. For Small loops, its position relative to the input image determines whether it is an Upper or Lower loop.

The algorithm determines the loop type by first separating the input image into three regions (upper, middle and lower). By definition, a Large loop must have its minimum y-coordinate located at the upper region and its maximum y-coordinate located at the lower region (note that y increases downwardly). If the loop is not a Large loop, another detection method is applied. In this case, the input image region is divided

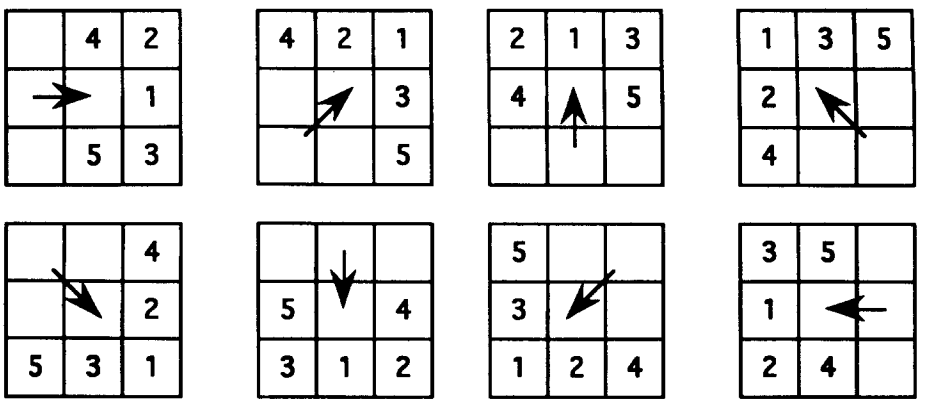


Fig. 8. Priority templates. The arrrow represents the previous direction. Priority 1 (High) to priority 5 (Low).

into an upper half and a lower half. The mid-point of the maximum and minimum y-coordinates of a loop is calculated and the loop is classified according to the mid-point location. The Large loop type is used to accurately determine the digit zero.

5.2.2. *Direct loop feature.* We have to detect a loop before positioning it. The simplest form of loop is that directly connected by only one feature point and is defined as a direct loop. In the 16×16 adjacency matrix, it can be directly detected by checking whether there are any connections from node i to node i. Once a direct loop is detected, its connection in the adjacency matrix will be deleted to prevent it from being detected again.

5.2.3. *Indirect loop feature.* Another form of loop is that which is connected by more than one node. In other words, the loop is composed of multiple segments. Two main problems are involved here:

(i) How to traverse the whole image to search for an indirect loop?
(ii) Which paths should be counted if multiple connections exist as parts of the loop?

The first problem can be solved by using a modified depth first search algorithm. The algorithm is similar to that of skeleton tracing but with the stack storing the matrix column indices which have more than one connection. Every time an element is popped from the stack the search process will start. The process terminates when the end of a branch is reached or a loop is detected. When starting from a node, every node reached is stored in a path list and the traced connection is removed. The node just reached is compared with the previous nodes in the path list. If that node is already in the previous path list an indirect loop is detected. The path for that loop is then transferred to another linked list data structure for further processing. When the search process reaches the end of a branch or a loop is detected, the top of the stack (another branch) is then popped to begin another search process.

After the paths of loops are recorded, the next step is to obtain the maximum and minimum coordinate values of the loops. For instance, the loop $(2 \rightarrow 3 \rightarrow 2)$ shown in Fig. 9 can be represented by the walks (a, b), (a, c) or (b, c). It is preferable to identify the loop that is not part of another loop, especially in digit recognition. Therefore, the desired walks for the above loop are (a, b) and (b, c). How to choose the loop walks in a multiple connection situation still remains to be answered.

To ensure that a loop will not contain another loop, the set of walks must be chosen such that the size is minimized. The proposed solution is to pick the next connection with the smallest length. In the example of Fig. 9, the algorithm will choose "b" (the smallest of the three connections), thereafter, it will choose "a" (assuming length of a < length of c) for the first indirect loop. To identify the second indirect loop it first goes back to node 3 through b and then goes through the only remaining connection, "c".

After the algorithm determines the walks for every loop, it calculates the maximum and minimum coordinates to determine the type of each loop, as in the case of a direct loop. Since the maximum and minimum coordinate values for all connections were already recorded during the skeleton tracing, the maximum and minimum coordinates of the loop can be obtained easily by comparing the size of the connections in loop walks.

5.3. *Curve and line extraction*

The primitives that are left in the 16×16 adjacency matrix after the loop extraction are straight and curved line segments. These line segments vary considerably in shape and length. A line segment may be composed of curves and straight lines. Moreover, the degree of straightness of lines and curvature of curves also vary with different inputs.

The method applied in this curve detection stage is an algorithm of cumulative angle change in feature detection, reported in reference (48). Furthermore, the corner-finding algorithm, reported in reference (49), is used to calculate the angle. The basic concept is the calculation of the curvature by the cumulative angle-change.

The angle-change is defined as the increment or decrement in the angle that adjacent straight lines make with the x-axis. In other words, it is the difference in angles between adjacent linear approximations. For a curve feature, the angle-change will always have the same sign. The sign of an angle-change is defined by whether it is a clockwise (positive) or anti-clockwise (negative) change. When another curve exists in the same line segment the sign of the angle-change will be reversed, as shown in Fig. 10.

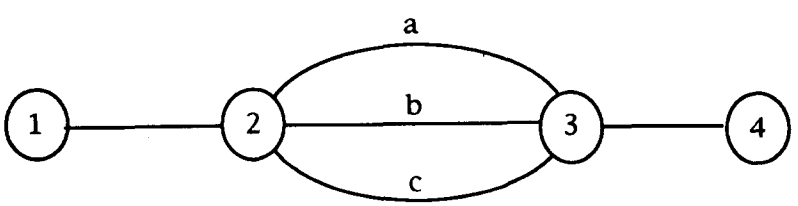


Fig. 9. Example for calculating loop coordinates.

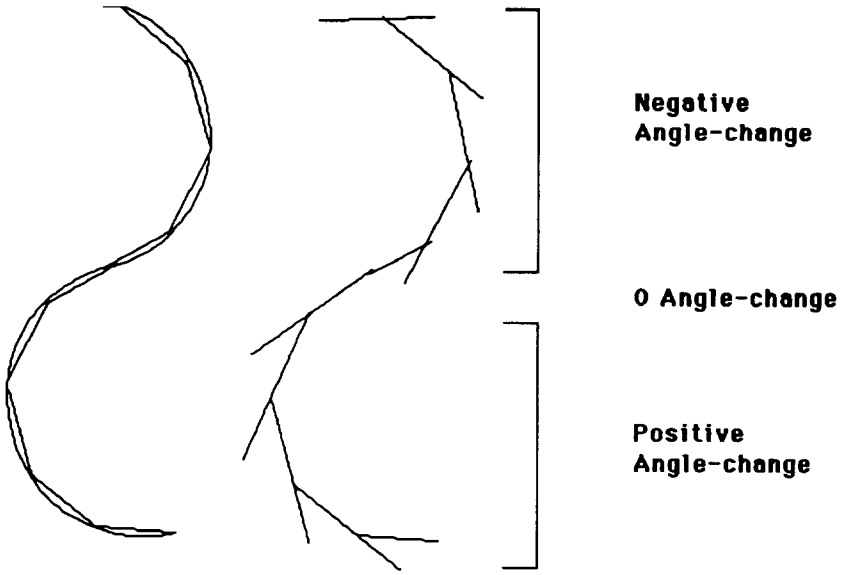


Fig. 10. Angle-change of an inverted S shape curve.

As for other features, the direction and position information of lines and curves is required for recognition. For a straight line feature, four directions are defined as follows:

Horizontal	Vertical	Positive	Negative
—	|	/	\

The directional types of the straight line are classified by the angle it makes with the x-axis. During the curve and line detection step, the start and end points are recorded at the same time. The slope of the straight line joining the two points is used for the angle calculation.

Four types of curve direction are also defined according to the curve's opening as follows:

East	South	West	North
⊂	∩	⊃	∪

A curve is categorized by examining the variation in coordinate values of the start, the mid and the end points of the curve. It is worth noting that in the input image x increases from left to right, while y increases from up to down. The following table demonstrates the curve categorization using the coordinates.

	East	South	West	North
x-value	start > mid < end	—	start < mid > end	—
y-value	—	start > mid < end	—	start < mid > end

The position information of line and curve features is similar to that of a loop feature. A line or curve is classified as Large when it extends from the upper region to the lower region. Otherwise it is classified as Small Upper or Small Lower according to its midpoint location.

6. CHARACTER CLASSIFICATION

Having extracted the primitives, it is now required to store them in some form and associate them with a character. Each pattern should uniquely identify a character and each character may be represented by several distinct patterns. These patterns and the characters they identify are fed to neural networks. The neural network is then trained to identify each character. After this, if the pattern for an unknown character is presented to the neural network, it would be able to classify the character based on the identical pattern (or that which matches closely) to which it has been trained.

The primitives will be classified using a feed-forward neural network trained by back-propagation.[50] Every Arabic character can be composed of at most five segments. A segment can be a complementary character, line, curve or loop. A 12-bit segment coding scheme is used to encode these segments. The scheme is depicted in Fig. 11.

The interpretation of the last seven bits of the code depends on the first five bits. However, only one of the first five should be 1. If zero or more than one bit is 1, the system automatically rejects the segment. For a segment to be misidentified, two bits would need to be incorrectly transmitted, including the one that identifies the segment. Therefore, the encoding is relatively robust.

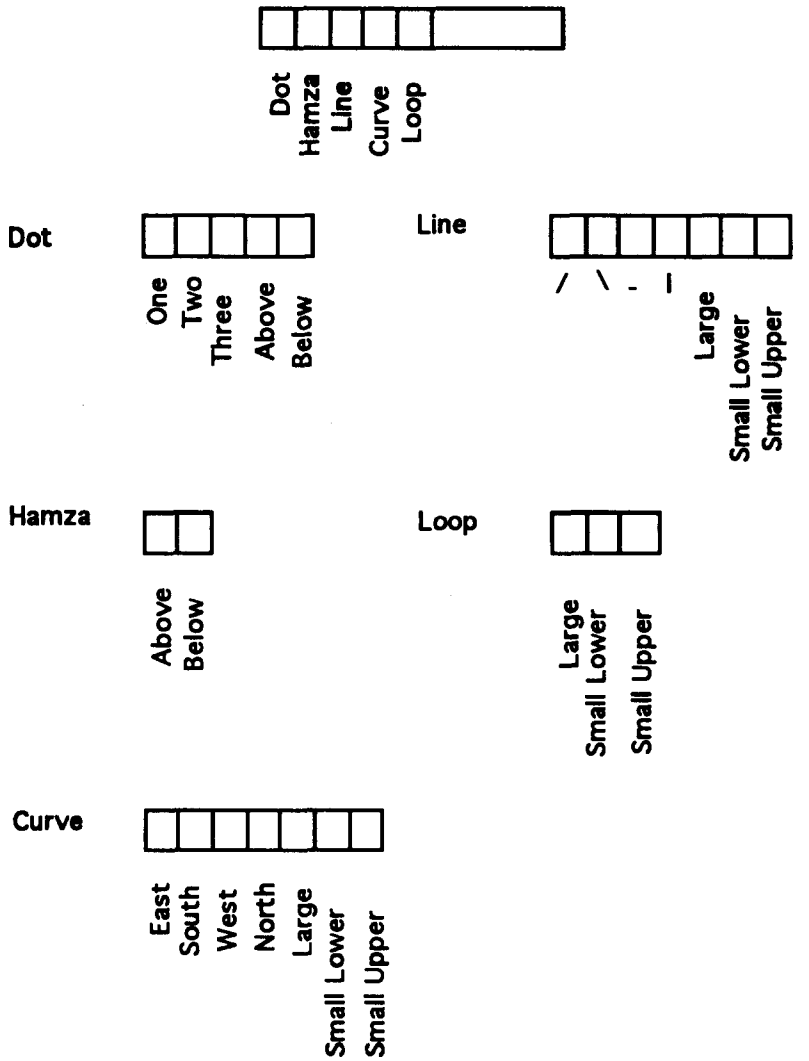


Fig. 11. Segment encoding for neural network input layer (each square cell holds a bit which acts as a flag).

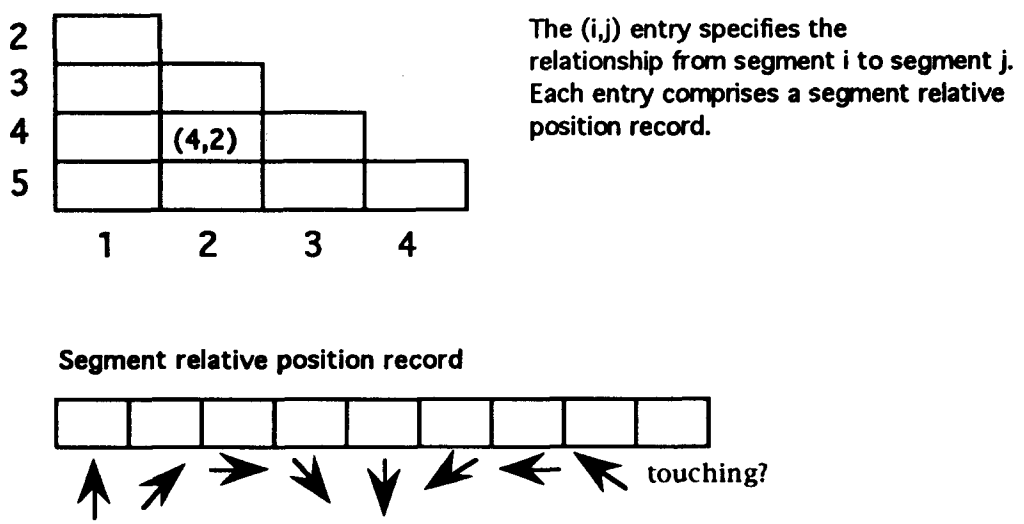
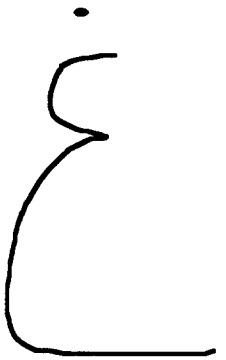


Fig. 12. Representation of segment interrelationships for the neural network input layer.



Segment 1 (1 Dot Above)
1 0 0 0 0 1 0 0 1 0 X X
Segment 2 (East Curve Small Upper)
0 0 0 1 0 1 0 0 0 0 0 1
Segment 3 (East Curve Large)
0 0 0 1 0 1 0 0 0 1 0 0

Segment inter-relationship matrix
(2,1) Relationship 1 0 0 0 0 0 0 0 0
(3,1) Relationship 1 0 0 0 0 0 0 0 0
(3,2) Relationship 1 0 0 0 0 0 0 0 1

Fig. 13. Complete representation of an Arabic character for the neural network input layer.

Table 1. Results for test data.

Size of the training set	Neural net architecture	Number of characters tested	Recognition rate	Misrecognition rate	Rejection rate
528 out of 2000	150 Input + 56 Hidden + 100 Output	1000	92%	5%	3%

The study uses a five-layer artificial neural network with one output unit for each possible character output. Each character will be classified in terms of the segments used in the system such as dot, hamza, line, curve or loop. The characteristics of each segment are determined in each case. The relationships between the segments are encoded in the *object inter-relationship matrix*, as depicted in Fig. 12. For each pair of distinct segments, this matrix records what direction one has to go from the first to reach the second and also whether or not the segments touch.

The overall design of the input layer uses a total of 150 input neurons. The segment encoding scheme uses 12 neurons for each segment, i.e. 60 neurons for the five segments. Furthermore, the segment inter-relationship matrix uses 90 neurons (10 entries with 9 neurons each). In general, for an n-object neural network there are $12n + 9n\,(n-1)/2$ neurons.

Figure 13 gives an example of the representation of a character using this input layer design.

When the network is trained, the output layer, in response to a familiar input pattern or one which resembles a familiar pattern, activates the neuron corresponding to this character classification.

7. EXPERIMENTAL RESULTS

The most crucial step in the project is feature extraction. The neural network classifies the characters on the basis of the features. Feature classification must be optimal in order to extract the distinct primitives and correctly identify the character. However, the hand-printed characters tended to have "hairs" that created problems in generalizing the primitives. Another problem encountered with feature extraction was that of fixing the direction of curves. Rules had to be formulated and fine-tuned during character testing.

Samples of hand-printed isolated Arabic characters were gathered. There are about 100 characters (an Arabic character might have up to four shapes depending on its relative position in the word, e.g. at the beginning, in the middle, at the end or isolated). The features extracted from some of these samples were sufficient to uniquely identify the character. The features extracted from the other samples failed to determine the correct identity of the character. However, most of these unrecognized characters were ambiguous to the human eye.

The neural network was trained initially with 2000 characters. This neural net was once again tested with another 1000 characters and the recognition rate was 92%. The results are tabulated as in Table 1.

The neural network training in each case took a lot of time. The process of training was repeated a number of times and the network that had the minimum value of tss (i.e. the best local minima) was used for testing.

The technique which we have adopted in this study is a hybrid approach between a purely structural approach and a classification test. All the algorithms outlined in the previous sections were implemented as C programs on a powerful MS-DOS microcomputer system connected to a 600 dpi scanner. The system was tested by 10 different users, ranging from acceptable to poor in quality.

The misrecognition rate is due to the complexity of the Arabic characters. In certain cases, the connectivity between the group of dots adds to the ambiguity of character recognition. For example, the characters in Fig. 2 only differ in the complementary character (dot); they can be incorrectly identified if these dots become connected and then eroded in the thinning process.

8. CONCLUSIONS

The technique adopted in this paper for recognizing hand-printed Arabic characters using Neural Networks combines a structural approach (based on structure primitives such as curves, straight lines and loops in similar manner to which human beings describe characters geometrically) and a classification test. This approach is efficient for feature extraction and recognition. As indicated by the experiments performed, the algorithm results in a 92% recognition rate.

The thinning of the binary image was used in order to produce a correct representation of the structure of the image by tracing the skeleton. However, the erosions experienced in the image as well as a large amount of variation likely in hand-written characters means that a machine learning technique to devise a classification algorithm might be a reasonable alternative. Such methods include symbol-processing-style algorithms such as 1D3 and C4.5 systems.[51]

In the area of recognition, a structural approach has been previously used. This approach is sufficient to deal with ambiguity without using contextual information. This area remains undeveloped due to the immaturity of vital computational principles for Arabic character recognition.

REFERENCES

1. I. Guyon, P. Albrecht, Y. Le Cun, J. Denker and W. Hubbard, Design of a neural network character recognizer for a touch terminal, *Pattern Recognition* **24**(2), 105–119 (1991).
2. Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard and W. Hubbard, Handwritten digit recognition: application of neural network chips and automatic learning, *IEEE Commun. Mag.* 41–46 (November 1989).
3. R. J. Williams and D. Zipser, A learning algorithm for continually. running fully recurrent neural networks, *Neural Comput.* **1**(2), 270–280 (1989).
4. J. Cao and M. Ahmadi, Statistical and neural classification of handwritten numerals: a comparative study, *11th ICPR*, 643–646 (1992).
5. P. J. G. Lisboa and S. J. Parantonis, Invariant digit recognition by Zernike moments and third-order Neural Networks, *2nd Int. Conf. Artific. Neural Networks* 82–85 (1991).
6. D. Burr, Neural net recognition of spoken and written text, *IEEE Trans. ASSP* **36**, 1162–1168 (1988).
7. A. Rajavelu and R. Musavi, A neural networks approach to character recognition, *Neural Networks* **2**, 387–394 (1989).
8. M. K. Brown and S. Ganapathy, Preprocessing technique for cursive script word recognition, *Pattern Recognition* **19**(1), 1–12 (1991).
9. R. H. Davis and J. Lyall, Recognition of handwritten characters—a review, *Image Vis. Comput.* **4**(4), 208–218 (1986).
10. E. Lecolinet and O. Baret, Cursive word recognition: methods and strategies, *Fundamentals in .Handwriting Recognition*, S. Impedovo, ed., pp. 235–263 (1994).
11. C. Y. Suen, R. Shingal and C. C. Kwan, Dispersion Factor: a quantitative measurement of the quality of handprinted characters, *Int Conf. Cybernet. Soc.* 681–685 (1977).
12. J. Kim and C. C. Tappert, Handwriting recognition accuracy versus tablet resoluton and sampling rate, *7th Int Conf. Pattern Recognition.* Montreal, pp. 917–918 (1984).
13. C. A. Higgins and R. Whitrow, On-line cursive script recognition, *Proc. Int Conf. Human-Comput. Interact. INTERACT'84* 139–143 (1985).
14. J. R. Ward and T. Kuklinski, A model for variability effects in handprinted with implication for the design of handwritten character recognition system, *IEEE Trans. Man Cybernet.* **18**, 438–451 (1988).
15. C. A. Higins and D. M. Ford, On-line Recognition of connected handwriting by segmentation and templet matching, *11th Int. Conf. Pattern Recognition* **II**, 200–203 (1992).
16. F. Nouboud and R. Plamondon, On-line recognition of handprinted characters: survey and beta tests, *Pattern Recogniton* **25**(9) 1031–1044 (1990).
17. C. Y. Suen, M. Berthod and S. Mori, Automatic recognition of handprinted characters, the state of the art, *Proc. IEEE* **68**(4), 469–483 (1980).
18. J. R. Ullmann, Adavance in character recognition, *Application of Pattern Recognition*, K. S. Fu, ed., pp. 197–236 (1982).
19. R. M. Bozinovic and S. N. Srihari, Off-line cursive script recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **11**(1), 655–663 (1989).
20. V. K. Govindan and A. P. Shivaprasad, Character recognition—a review, *Pattern Recognition* **23**(7), 671–683 (1990).
21. S. Impedovo, L. Ottaviano and S. Occhinegro, Optical character recognition—a survey, *Int. J. Pattern Recognition Artific. Intell.* **5**(1/2), 1–24 (1991).
22. A. Amin, A. Kaced, J. P. Haton and R. Mohr, Handwritten Arabic characters recognition by the IRAC system, *5th Int. Conf. Pattern Recognition* pp. 729–731. Miami, U.S.A. (1980).
23. A. Amin, Machine recognition of handwritten Arabic word by the IRAC II system, *6th Int Conf. Pattern Recognition* pp. 34–36. Munich (1982).
24. E. M. Riseman and R. W. Ehrich, Contextual word recognition using binary diagrams, *IEEE Trans. Comput.* **c-20**(4), 397–403 (1971).
25. A. Amin, G. Masini and J. P. Haton, Recognition of handwritten Arabic words and sentences, *7th Int. Conf. Pattern Recognition*, pp. 1055–1057. Montreal (1984).
26. T. S. El-Sheikh and S. G. El-Taweel, Real-time Arabic handwritten character recognition, *Pattern Recognition* **23**(12), 1323–1332 (1990).
27. S. Al-Emami and M. Usher, On-Line recognition of handwritten Arabic characters, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-12**, 704–710 (1990).
28. A. Belaid and J. P. Haton, A Syntactic approach for handwritten mathematical formula recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**, 105–111 (1984).
29. B. Parhami and M. Taraghi, Automatic recognition of printed Farsi texts, *Pattern Recognition* **14**(6), 395–403 (1981).
30. S. Saadallah and S. Yacu, Design of an Arabic character reading machine, *Proc. Comput. Process. Arabic Language.* Kuwait (1985).
31. A. Amin and G. Masini, Machine recognition of multifonts printed Arabic texts, *8th Int. Conf. Pattern Recognition* pp. 392–395. Paris (1986).
32. A. Amin and J. F. Mari, Machine recognition and correc-

tion of printed Arabic text, *IEEE Trans. Man Cybernet.* **9**(1), 1300–1306 (1989).
33. H. Freeman, On the encoding of arbitrary geometric configuration, *IEEE Trans. Electr. Comput.* **EC-10**, 260–268 (1968).
34. D. Forney, The Viterbi algorithm, *Proc. IEEE* **61**(3), 268–278 (1973).
35. H. Almuallim and S. Yamaguchi, A method of recognition of Arabic cursive handwriting, *IEEE Trans. Patt. Anal. Mach. Intell.* **PAMI-9**, 715–722 (1987).
36. T. El-Sheikh and R. Guindi, Computer recognition of Arabic cursive script, *Pattern Recognition* **21**(4), 293–302 (1988).
37. F. El-Khaly and M. Sid-Ahmed, Machine recognition of optically captured machine printed Arabic text, *Pattern Recognition* **23**(11), 1207–1214 (1990).
38. S. El-Dabi, R. Ramsis and A. Kamel, Arabic character recognition system: Statistical approach for recognizing cursive typewritten text, *Pattern Recognition* **23**(5), 485–495 (1990).
39. H. Al-Yousefi and S. S. Udpa, Recognition of Arabic characters, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-14**, 853–857 (1992).
40. A. Shoukry, A sequential algorithm for the segmentation of typewritten Arabic digitized text, *Arabian J. Sci. Engng* **16**(4), 543–556 (1991).
41. T. Wakayama, A core-line tracing algorithm based on maximal square moving, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-4**, 68–74 (1982).
42. V. Margner, SARAT—A system for the recognition of Arabic printed text, *11th Int Conf. Pattern Recognition.* **II**, 561–564 (1992).
43. H. Al-Sadoun and A. Amin, A new structural technique for recognizing printed Arabic text, *Int J. Pattern Recognition Artific. Intell.* **9**(1), 101–125 (1995).
44. N. Otsu, A threshold selection method from gray level histogram, *IEEE Trans. Syst. Man. Cybernet.* **SMC-9**(1), 62–66 (1979).
45. J. S. Weszka and A. Rosenfeld, Histogram modification from threshold selection, *IEEE Trans. Syst. Man Cybernet.* **SMC-9**(1), 38–52 (1979).
46. A. Amin and H. B. Al-Sadoun, Hand printed Arabic character recognition system, *12th Int Conf. Pattern Recognition* 536–539 (1994).
47. B. K. Jang and R. T. Chin, One pass parallel thinning: analysis, properties and quantitative evaluation, *IEEE PAMI* **14**(11), 1129–1140 (1992).
48. C. Nadal, R. Legault and C. Y. Suen, Complementary algorithms for the recognition of totally unconstrained handwritten numerals, *Pattern Recognition Int Conf.* 443–449, (1990).
49. H. Freeman and L. S. Davis, A corner algorithm for chain-coded curves, *IEEE Trans. Comput.* 297–303 (1977).
50. D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Parallel Distributed Processing*, Vol. 1. Bradford Books, MIT Press, Massachussetts (1986).
51. J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kauffman, San Mateo CA, (1993).

About the Author—ADNAN AMIN received the B.Sc. degree in mathematics and Diploma in computer science from Baghdad University in 1970 and 1973, respectively. D. E. A. in electronics from the University of Paris XI (Orsay) in 1978 and presented his Doctorate D'Etat in 1985 from the University of Nancy I (CRIN), France. From 1981 to 1985, he was Maitre Assistant at the University of Nancy II. From 1985 to 1987, he worked in INTEGRO (Paris) as Head of Pattern Recognition. From 1987 to 1990, he was Assistant Professor at Kuwait University. In 1991, he joined the School of Computer Science and Engineering at the University of New South Wales, Australia as a Senior Lecturer. His research interests relate to pattern recognition and artificial intelligence (document Image understanding ranging from character/word recognition to integrating visual and linguistic information in document composition), neural networks and machine learning.

About the Author—HUMOUD B. AL-SADOUN received the B.S.E.E. and M.S.E.E. degrees from the University of Wisconsin, Madison, in May 1977 and December 1977, respectively, the M.S.E. degree in Industrial and Operational Engineering from the University of Michigan, Ann Arbor in 1982 and the Ph.D. degree in electrical and computer engineering from the University of Michigan in 1985. From 1978 to 1979, He worked as an Instrumentation Engineer with the Kuwait National Petroleum Company. He is currently with the faculty of Electrical and Computer Engineering Department of Kuwait University. His research interests include pattern recognition, computer architecture and arabization techniques.

About the Author—STEPHEN FISCHER received his B.A. in mathematics and geophysics in 1989 from Rice University. He is currently a postgraduate student at the University of New South Wales. His main fields of research are document analysis and character recognition.